XrdHTTP
HTTP and WebDAV for the Xrootd framework

Fabrizio Furano

CERN IT-SDC/ID

The XrdHTTP plugin

- XrdHTTP gives pragmatic HTTP(s) and WebDAV support to the XRootD framework
- Goal: take an already existing XRootD-based storage cluster and add the HTTP protocol
- Easy and cheap, no new HW required, no new daemons, no gateways, no scripts, no glue

- **HTTP/WebDAV for XrootD is done**
  - Support basic HTTP, DAV with a few DM extensions (replicas etc.)
  - Plug into the XRootD framework to rely on its features (e.g. tapes or monitoring) without reconfiguring the site or installing complex stuff
  - Can share the same port of the XRootD protocol. The client is automatically detected
  - Supports X509 auth, proxy certificates and VOMS extensions (*)
  - Any XRootD server will keep its advanced functionalities, PLUS HTTP compliance
  - Works on XRootD4, IPv6 compatible from the start
  - Expected to work on EOS
  - Done preliminary testing with Rucio, some minor fixes are done.

(*) Through an external plugin

Why HTTP/DAV?

- Interesting technical features
  - Multitalented, covers most existing use cases, while allowing new stuff
  - Applications just go straight to the data, wherever they are running
  - Staging (GET/PUT) or direct chunked access
  - Supports WAN direct access

- It's there, whatever platform we consider
  - HTTP is moving much more data than HEP worldwide, although in different ways
- We like browsers, they give a feeling of simplicity
  - Making people more open towards that technology
- Goes towards convergence
  - Users can use their preferred devices and apps to access their data
  - Sophisticated custom applications are allowed
  - Can more easily be connected to commercial systems and apps
- Attractive for a professional to be formed in these systems

HTTP and Grid computing

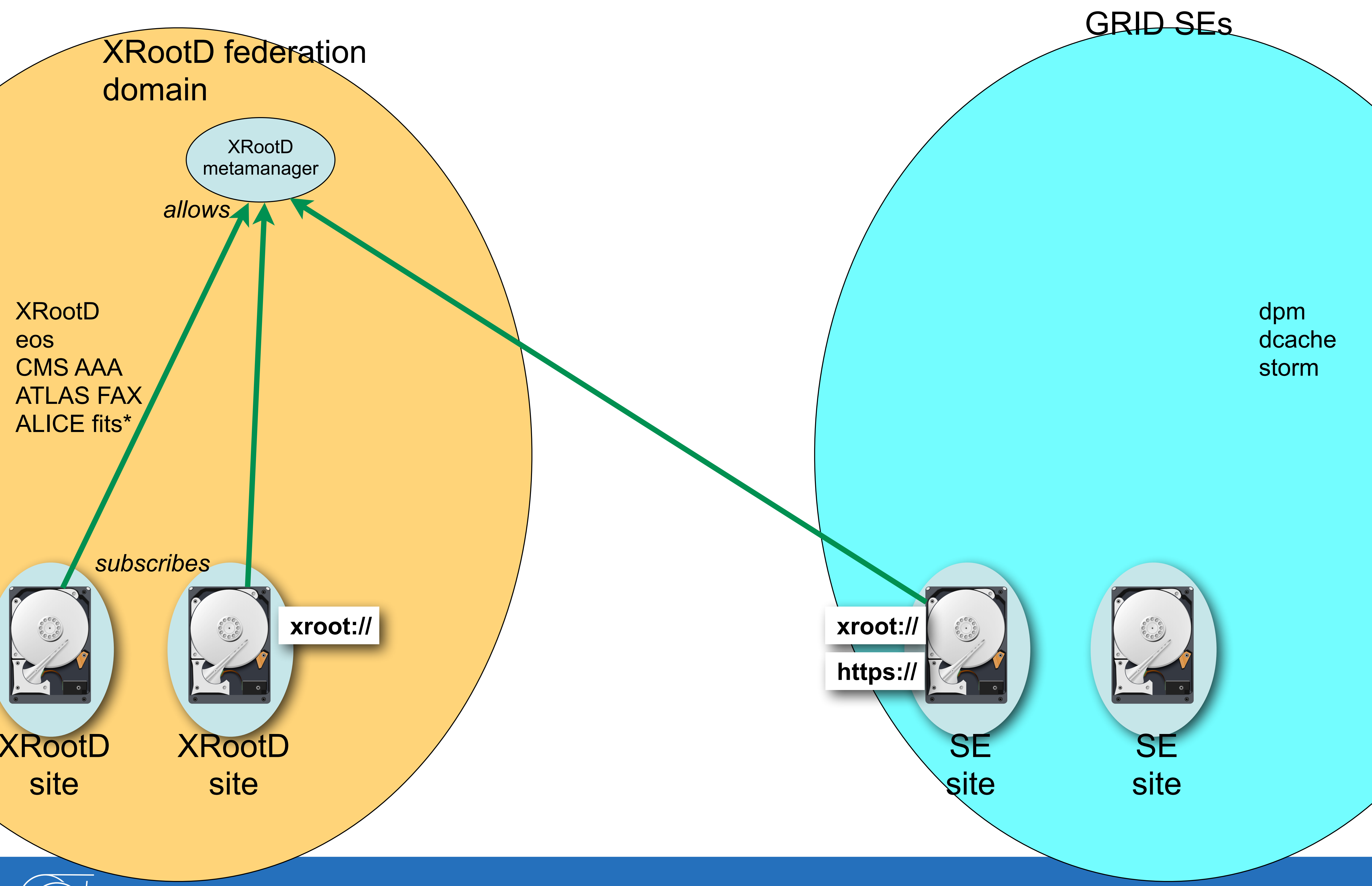
- We see potential in enabling HTTP and WebDAV for Grid data access

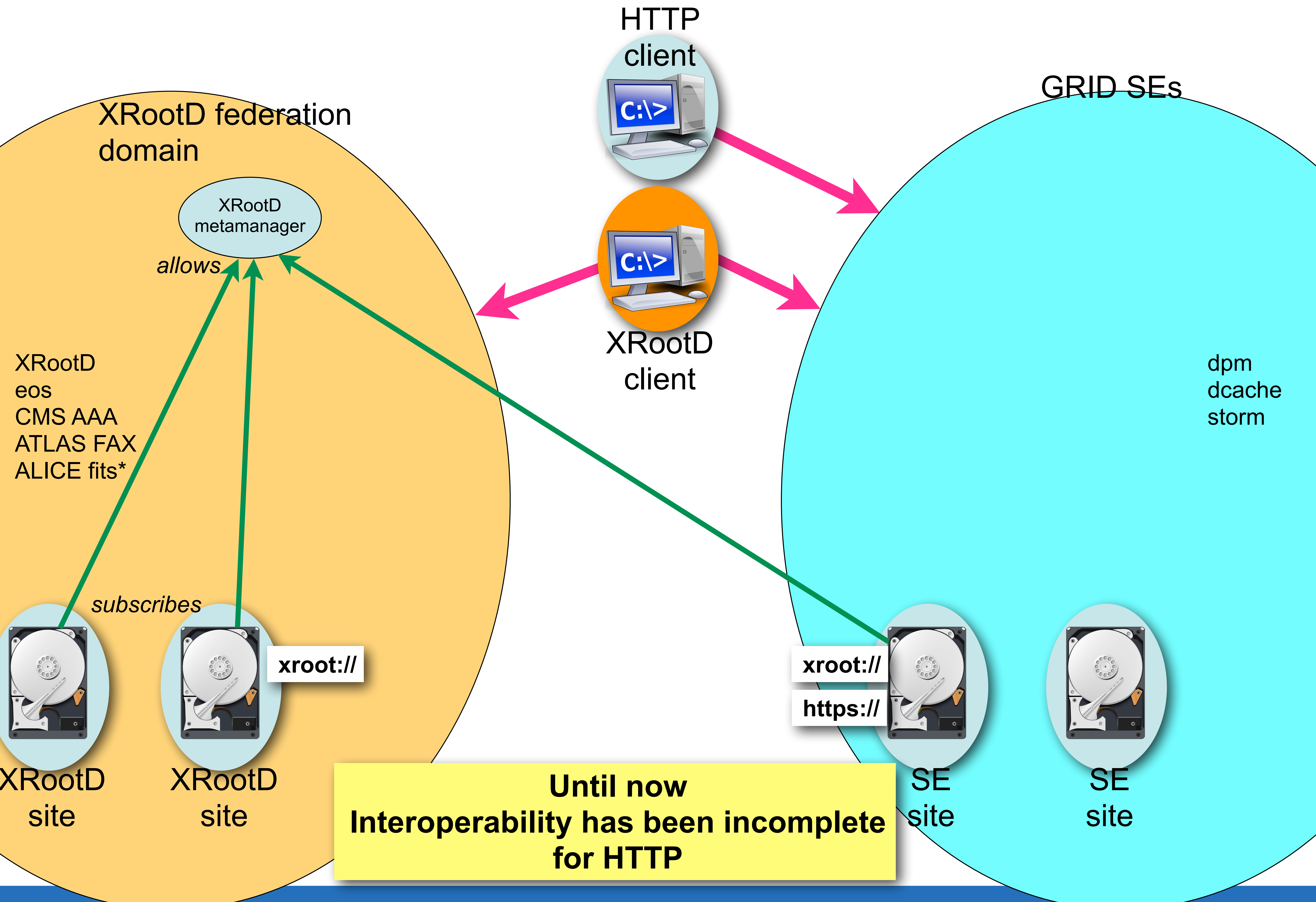
- For example, we may like to:
  - let browsers interact with Grid storage
  - let people use simple mainstream Web clients to do simple things
  - let people use advanced clients to get the full spectrum of features
  - let anyone write a Web interface (Java, Javascript) that interfaces Grid storage natively
  - Build integrations with Cloud services at the protocol level

HTTP and XRootD

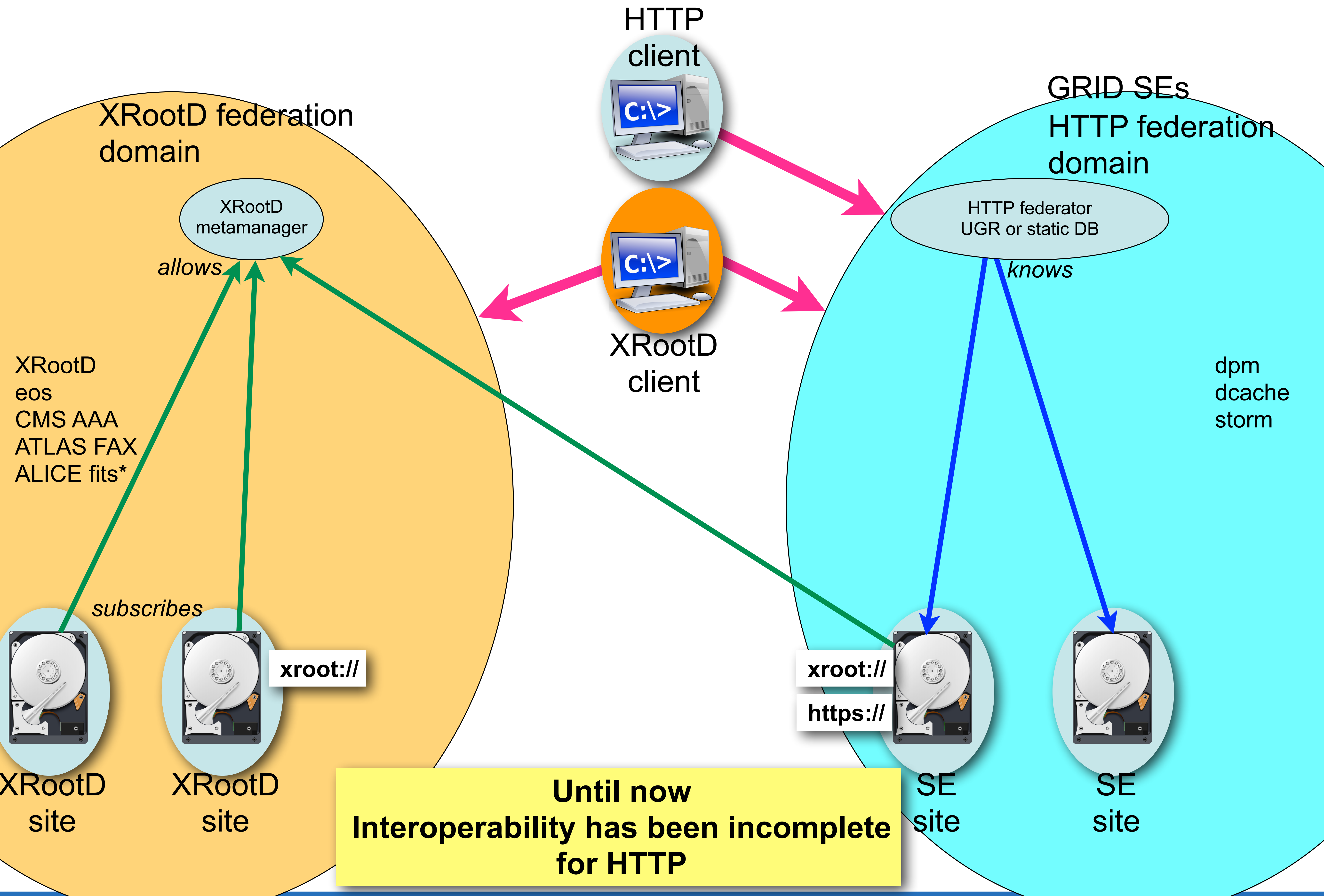
- Many instances of XRootD, not only HEP
  - Generally high quality deployments of a high quality framework
  - Sometimes very important ones (e.g. CERN with EOS)
  - Advanced features: tapes, data movements, monitoring, etc.
  - This involves also many other components and groups of people using those hooks
    - Monitoring of the FAX is a perfect example
- Many instances of http/dav compliant SEs in the Grid
- Once a site/organization has chosen a framework, changing it can be very expensive
  - Also quality of service is at risk
- Risk is having islands of protocols within the same community

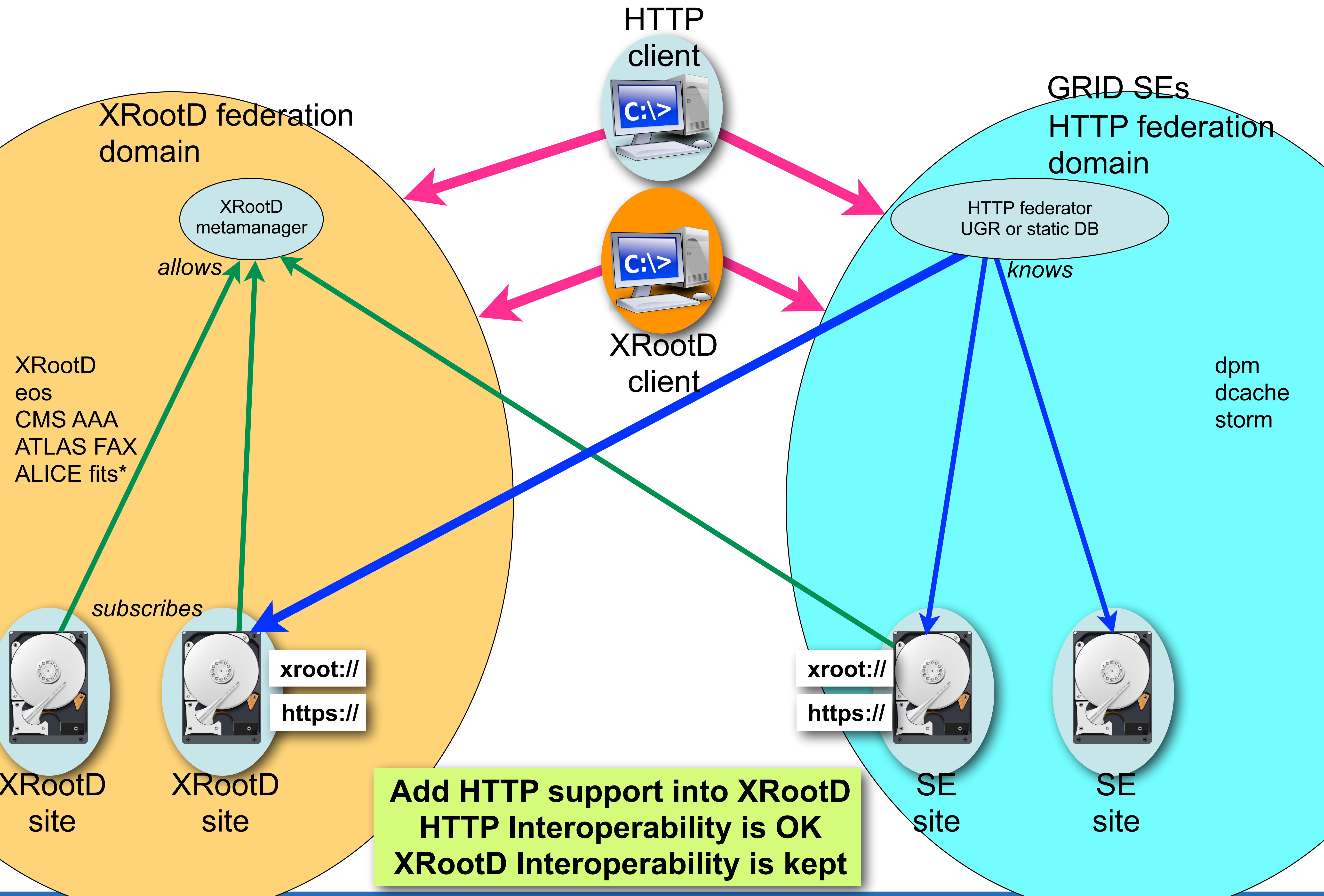
- How to fit XRootD-based SEs in a vision that allows HTTP ?

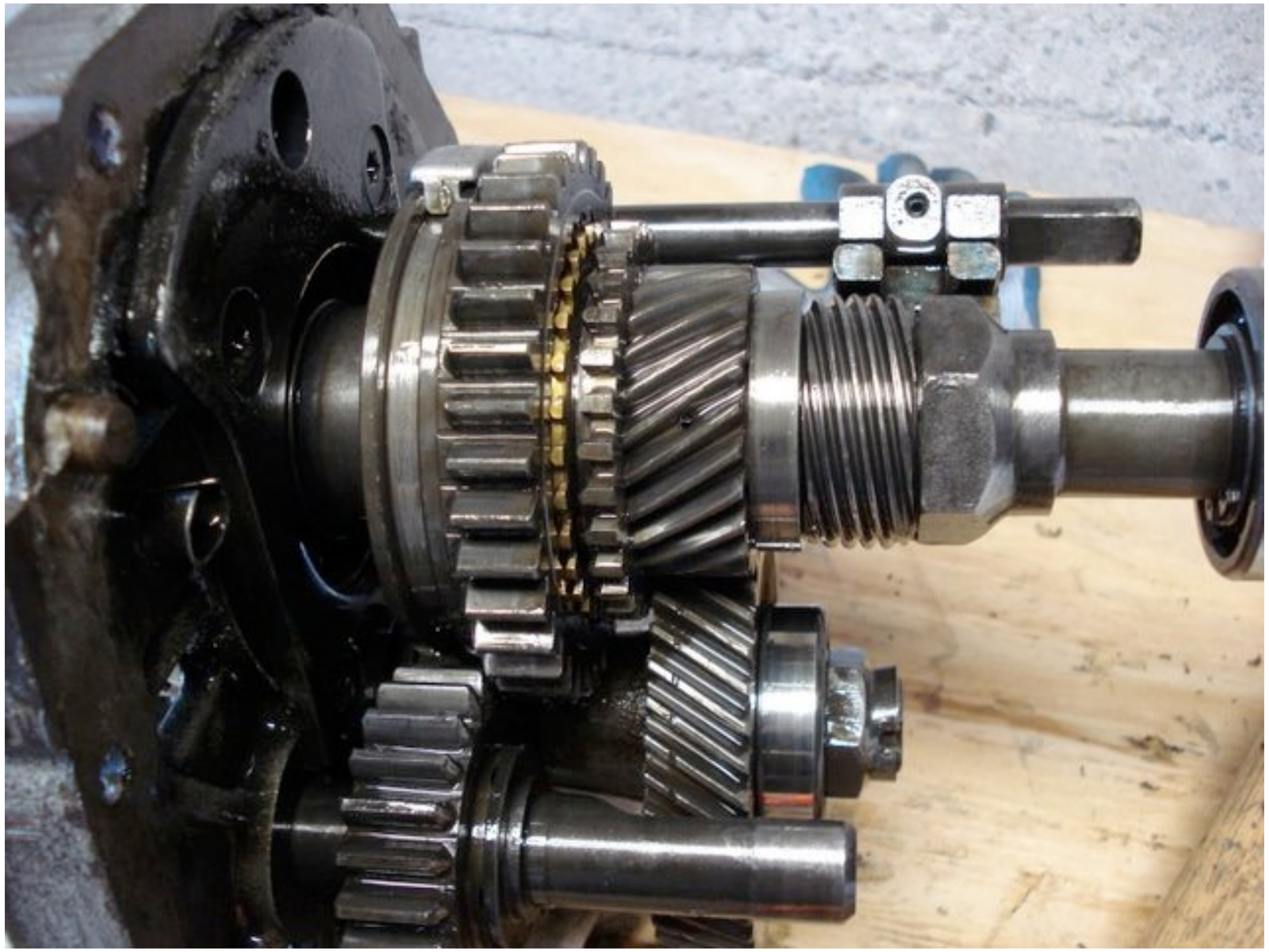

GRID SEs
XRootD federation
domain
XRootD
metamanager
allows
XRootD
eos
CMS AAA
ATLAS FAX
ALICE fits*
dpm
dcache
storm
subscribes
xroot://
xroot://
https://
XRootD
site
XRootD
site
SE
site
SE
site

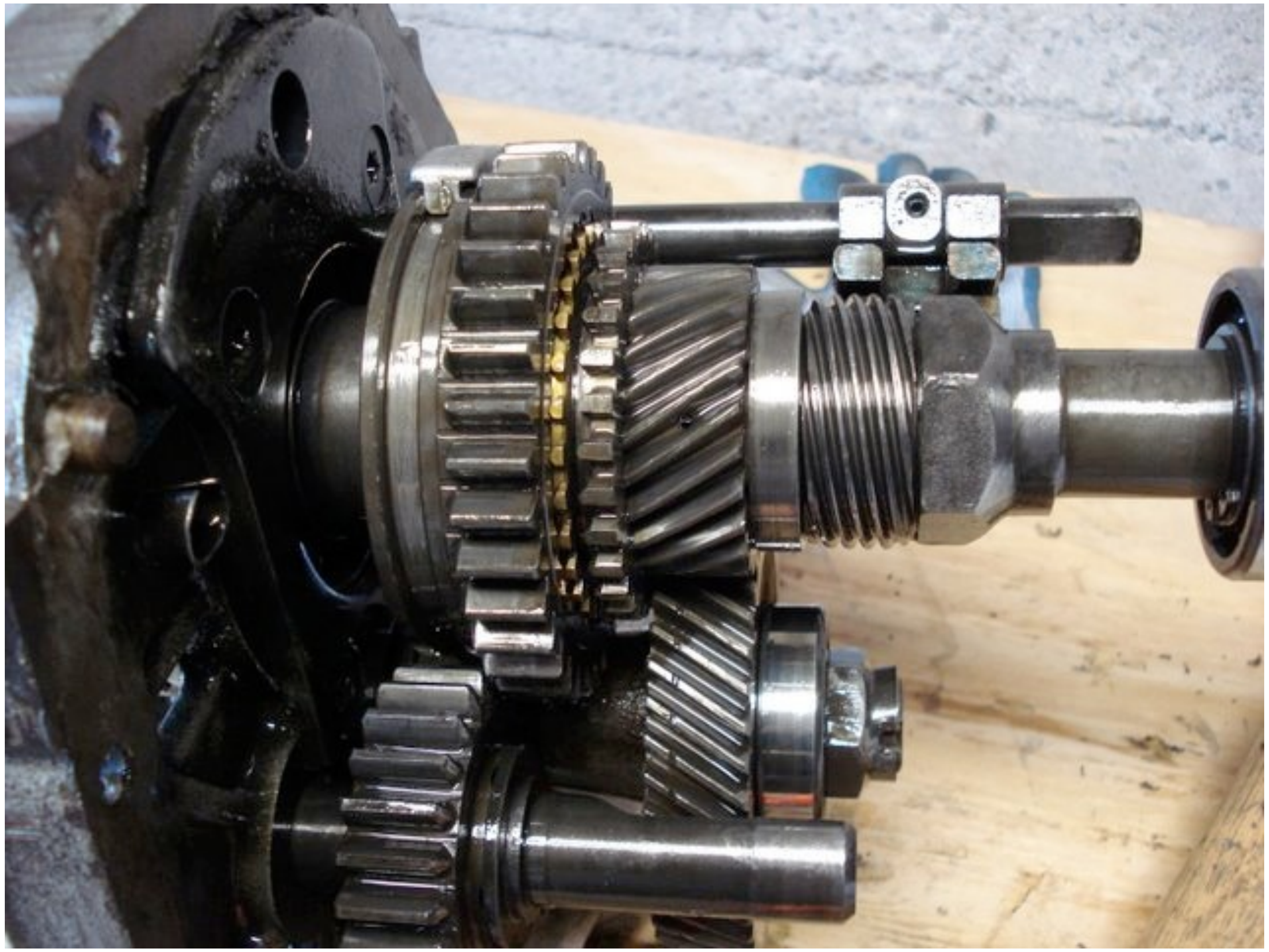


Until now
Interoperability has been incomplete for HTTP


HTTP client
XRootD federation domain
XRootD metamanager
allows
XRootD
eos
CMS AAA
ATLAS FAX
ALICE fits*
subscribes
xroot://
XRootD site
XRootD site
XRootD client
GRID SEs
HTTP federation domain
HTTP federator
UGR or static DB
knows
dpm
dcache
storm
xroot://
https://
SE site
SE site
Until now
Interoperability has been incomplete for HTTP


HTTP
client
XRootD federation
domain
GRID SEs
HTTP federation
domain
XRootD
metamanager
HTTP federator
UGR or static DB
allows
knows
XRootD
client
XRootD
eos
CMS AAA
ATLAS FAX
ALICE fits*
dpm
dcache
storm
subscribes
xroot://
https://
xroot://
https://
XRootD
site
XRootD
site
SE
site
SE
site
Add HTTP support into XRootD
HTTP Interoperability is OK
XRootD Interoperability is kept





How XrdHTTP works
Tech features explained

Supported primitives

- GET for files
- GET for dirs (HTML/CSS rendering)
- PUT
- PROPFIND (only at depth 1)
- HEAD
- OPTIONS
- MKCOL
- MOVE
- DELETE

GET and PUT: Chunk access

- GET supports Range headers
  - Single range to read scattered chunks one at a time
  - Multiple range, equivalent to a Vectored Read
  - ROOT TTreeCache works with XrdHTTP
    - Beware: TWebFile has issues, you must use TDavixFile (ROOT>5.34)

- PUT can only write whole files
  - No standard definition of ranges for PUT
  - Chunked write is not possible with PUT
  - Chunked write is called PATCH according to WebDAV
    - PATCH is not yet supported by XrdHTTP (nor by any server or client that I know)

Adding a protocol to Xrootd

- Always been possible to load different protocols implemented in so libraries e.g.
  - *XrdXrootd* --> gives what we call Xrootd
  - *XrdXProofd* --> gives the heart of PROOF
  - *XrdHTTP* --> gives HTTP and WebDAV

- The file access protocol *XrdXrootd* implements many sophisticated things: no-compromise I/O, monitoring, HSM, tape hooks, etc.
  - More than just the protocol semantics implementation
  - It's also the implementation of the features, i.e. high performance disk access, based on the low level functions of the Xrootd framework

- How to implement a new file access protocol (HTTP/DAV) without duplicating these difficult things?

The Xrootd Protocol Bridge

- *XrdBridge* allows to submit requests through memory to an *XrdXrootd* instance in use by the framework
  - The responses come through sync callbacks in the same process
  - This is a sort of internal in-memory gateway, giving xrootd protocol features to in-memory objects

- Hence the *XrdHttp* internal workflow is:
  - Manage the connection, detection, ssl, etc.
  - get the HTTP/DAV request header
  - get the security info from the connection (SSL, x509)
  - ask for more security info to specialized info extractors (VOMS)
  - translate HTTP/WebDAV requests into sequences of Xrootd requests
  - login into the Bridge passing the client's security credentials
  - inject the request into the *XrdBridge*
  - let *XrdBridge* handle autonomously the data chunks on the socket (= performance!)
  - collect the *XrdBridge* callback responses into a "response status"
  - Submit other partial *XrdBridge* requests OR craft the HTTP response (based on the current status of the request)

Xrd/HTTP/HTTPS Protocol detection

- We can configure XrdHTTP on any port, including the same port 1094 shared with the Xrootd protocol
  - **Hence, no need to reconfigure firewalls to add XrdHTTP**

- The Xrootd framework can run multiple protocols in the same TCP port
- The protocol implementations must be able to recognize their clients
- Note: HTTP and HTTPS live in the same port. Once a connection is given to XrdHTTP, it applies some heuristics to discriminate between http/https
- Basically "if it's not an ASCII HTTP request then try with SSL, otherwise fail"
- Works well!

Performance on header parsing (1/2)

- HTTP headers have not been designed with performance in mind. Well known story that will be fixed by http2
- XrdHTTP was designed to maximize the efficiency in reading/parsing the HTTP request headers
- Still it requires parsing, using a bit more CPU than the xrootd protocol
- Header ends with a double CRLF
  - Historical annoyance for performance
  - The problem: How to read from a stream, efficiently looking for a double CRLF
- The lazy student's solution… get 1 char at a time from the socket? Serious?
  - That would give internal latency and horrible CPU waste

Performance on header parsing (2/2)

- The XrdHTTP recipe:
  - Read as many bytes as possible from the socket in chunks as large as possible
    - Minimize the calls to read() and poll()
  - A circular buffer (hundreds of Ks) with efficient "readline" primitives
    - Virtually unlimited max header size (good for composite ops or crazy headers, cookies, etc.)
    - Max line length is the whole buffer
    - Allows code to be pedantic against buffer overflow attacks
  - If the buffer has read from the link past the double CRLF, *XrdBridge* can be injected (if needed) the remainder data read that is not part of the already processed header

Clustered setup

- Load XrdHTTP in a data server and we have an HTTP/DAV endpoint
- Load XrdHTTP in an xrootd manager and we have an XrdHTTP redirector
- A redirector follows the regular Xrootd redirection semantics,
  - Redirectors will try to redirect clients on all the primitives
  - Hence, client apps need to coherently process redirections for all the HTTP/DAV primitives to support XrdHTTP clustered operations
    - (Or other objects have to act as HTTP gateways)

Client support

- *libCurl, wget, libneon, cadaver,* etc…
  - These all work according to their limits
    - E.g. In the case of sophisticated things the app may have to interpret the redirect responses that they would return unprocessed
  - The typical cases of simple access work well
- Browsers just work, listings are rendered to HTML
- The client that we contributed is Davix
  - Fedora, EPEL, Solaris, Debian, Windows
  - http://dmc.web.cern.ch/projects/davix/home
- TDavixFile on ROOT 5.34 and ROOT 6
  - About to appear in the LCG releases
- Everyone if free to design some fancy HTML/Java/ Javascript interface

XrdHTTP Basic Configuration

- All the flavors with very few statements
- Basic server with HTTP/HTTPS/DAV/DAVS:

```
if exec xrootd
   xrd.protocol XrdHttp /usr/lib64/libXrdHttp.so.1
fi

# Drop these for an open plain HTTP/DAV server
http.cert /etc/grid-security/hostcert.pem
http.key /etc/grid-security/hostkey.pem
http.cadir /etc/grid-security/certificates

... inserted into an existing config file
```

- Will work also for redirectors
- We can do better though, avoiding that HTTPS clients do the sec handshake twice
  - Once with the redirector + once with the data server
- Moreover, HTTPS data servers will be slower

Clustered security configuration (1/2)

- *plain http://* ... everything unencrypted, no security
- *full https://* ... secure auth, secure redir, secure data xfers
  - The default when certs are configured
  - HTTPS handshakes happen in both redirector and data server

- *https-to-http with security token* :
  - secure authentication, HTTPS->HTTP redir to data server with encrypted security token, http data xfers
  - HTTPS handshakes will happen in the redirector only

- *http-to-https*
  - HTTP->HTTPS to secure data servers
  - HTTPS handshakes will happen in the data server only (= distributed)
  - HTTPS data transfers in the data servers ... may mean higher load and more latency

- *http-to-https-to-http with security token*
  - HTTP->HTTPS redir, HTTPS auth, HTTPS->HTTP redir to the same host with security token, plain data xfers
  - The HTTPS handshakes are distributed in the data servers

Clustered security configuration (2/2)

- Configuring certs and CAs tells to a server that it has to support https
- Tell if the redir destination is http or https

```
# If this server needs to redirect, we may want to choose if
# to redirect with http or https
http.desthttps no|yes
```

- Simple Shared Secret encrypted tokens to carry auth information through plain HTTP redirection

```
# The key that has to be shared between HTTP redirectors and servers in this
#  cluster. The minimum length is 32 characters, hence the default
#  one will not work.
# A data server with the secret key set will only accept
#  - requests that have been correctly hashed
#  OR
#  - requests using https (if https is configured)
http.secretkey CHANGEME_MINIMUM32CHARS
```

VOMS support

- XrdHTTP implements HTTPS authentication and X509
- It extracts auth data from the SSL connection and passes it to the Xrd framework for normal authorization processing
- XrdHTTP can load an additional “Security Extractor” plugin that can amend this information before it is submitted to the rest of the Xrd framework
- We wrote a VOMS security extractor plugin
- No other config beside loading it

```
http.secxtractor /usr/lib/libXrdHttpVOMS.so
```

- NOTE: due to the dependency towards libVoms2, this plugin is not shipped with XrdHTTP inside the Xrootd sources
- We will make it available when Xrootd4 is released in EPEL

Clusters, clients and file listings (1/2)

- An xrootd data server can only provide the listings for the files it contains
- An xrootd redirector does not provide any listings, just redirections!

- The native xrootd client works around this, and to collect a listing it crawls the cluster, querying all the servers
- A generic HTTP or DAV client cannot do that, the listing must come from a single place

- Web browsers are implicitly connected to the idea of listings. Not having them would be very questionable!
- Web browsers mean interactivity. The listings must be quick or the user experience will be disappointing.

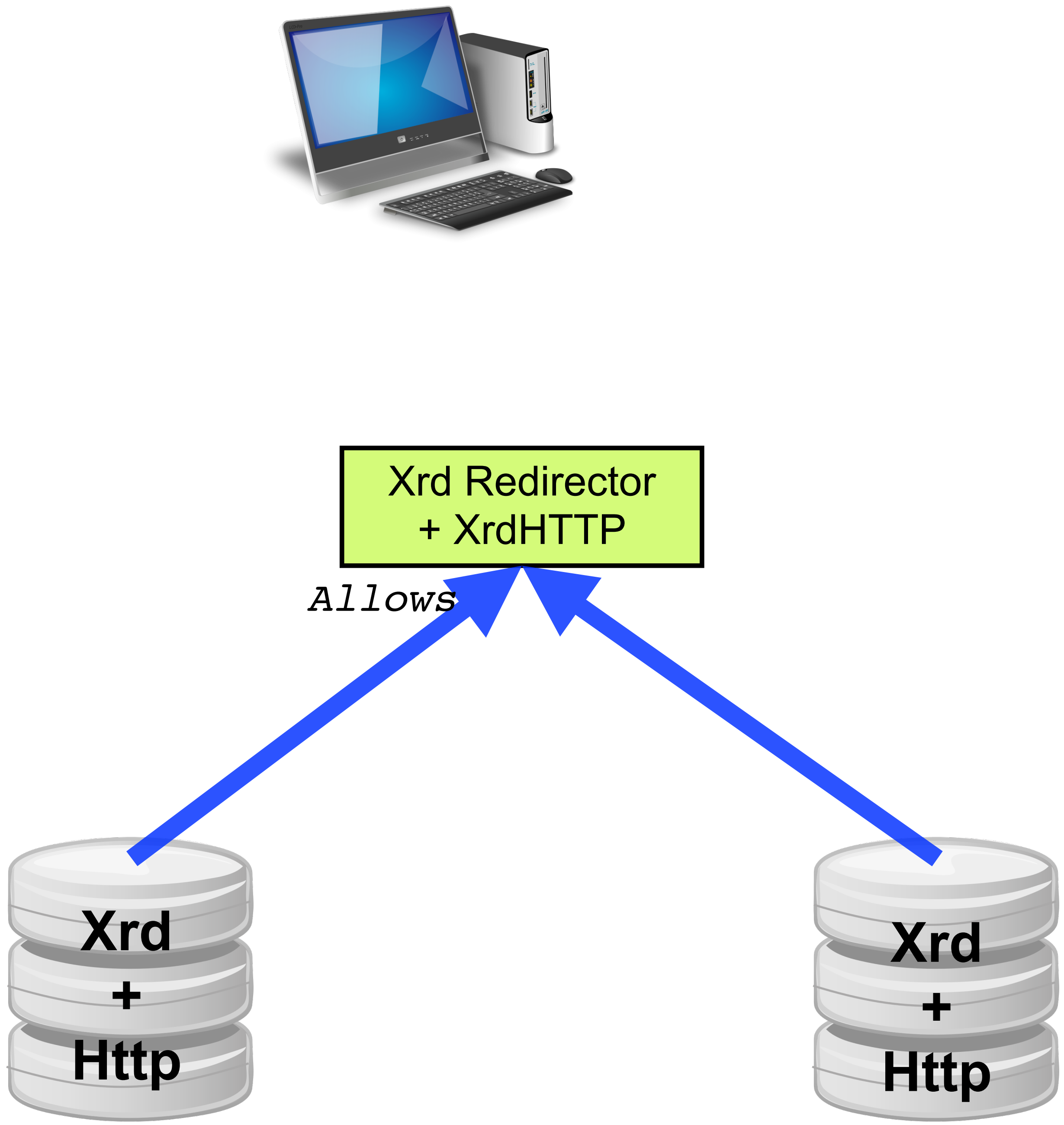
- A single XrdHTTP data server will just work.
- A cluster with XrdHTTP support can redirect PROPFIND and GET (for directories) to an external system that knows how to provide quick listings

```
# Redirect on listing request. Client must support redirs!
http.listingredir http://mynewhostwhichprovideslistings:80/
```

- then… how to provide quick listings for many users and vast metadata repos?

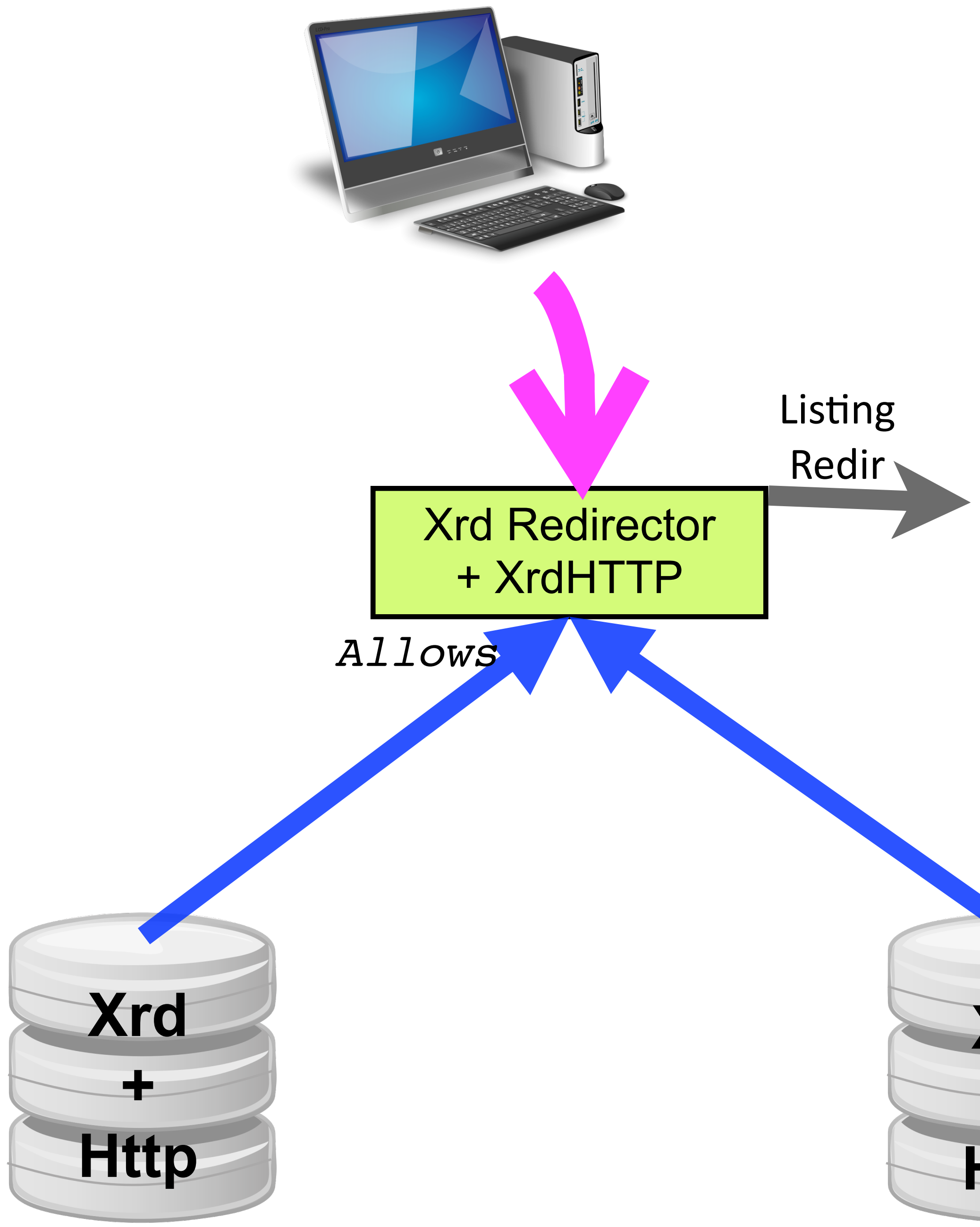
Clusters, clients and file listings (2/2)

- We have successfully evaluated the Dynamic Federations to provide those listings for a cluster
  - Speed and scalability in collecting metadata on-the-fly
  - See the other presentation on the Dynafeds and the Ugr (Uniform Generic Redirector)



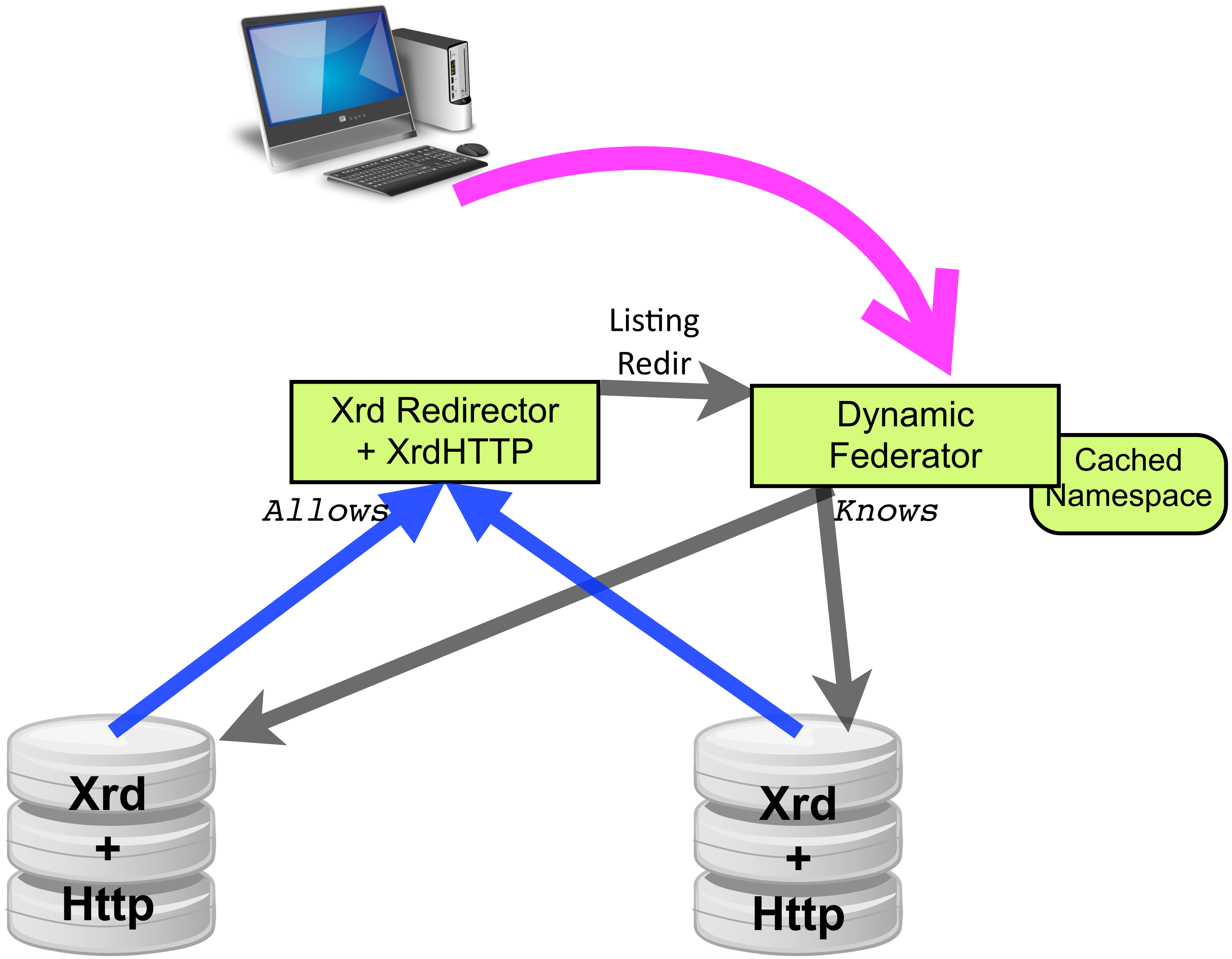
Clusters, clients and file listings (2/2)

- We have successfully evaluated the Dynamic Federations to provide those listings for a cluster
  - Speed and scalability in collecting metadata on-the-fly
  - See the other presentation on the Dynafeds and the Ugr (Uniform Generic Redirector)



Clusters, clients and file listings (2/2)

- We have successfully evaluated the Dynamic Federations to provide those listings for a cluster
  - Speed and scalability in collecting metadata on-the-fly
  - See the other presentation on the Dynafeds and the Ugr (Uniform Generic Redirector)



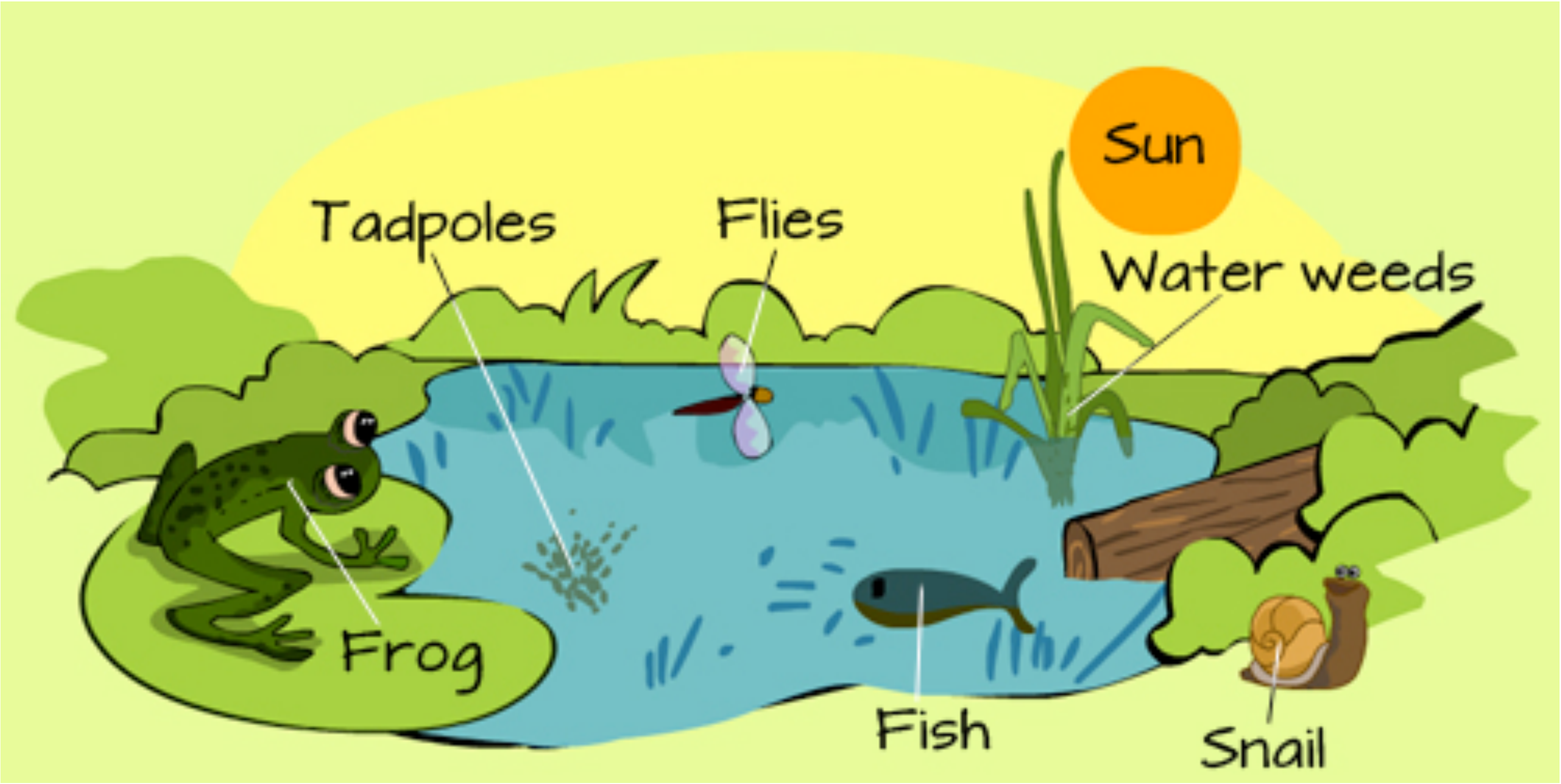
The HTTP ecosystem for HEP

- **DPM**
- dCache
- STORM
- **HTTP for XRootD**
- **HTTP Federations**
- **FTS3**
- **DAVIX**
- **ROOT with TDavixFile**

- We want seamless interoperability, which is more than coexistence
- Performance and ability to compose services
- Any standard client will work and give its features
- Our contribution “fills the gaps” between plain Web and HEP data access
- Browsers are supported
- An advanced client will give all the features (implemented using standards)

The HTTP ecosystem for HEP

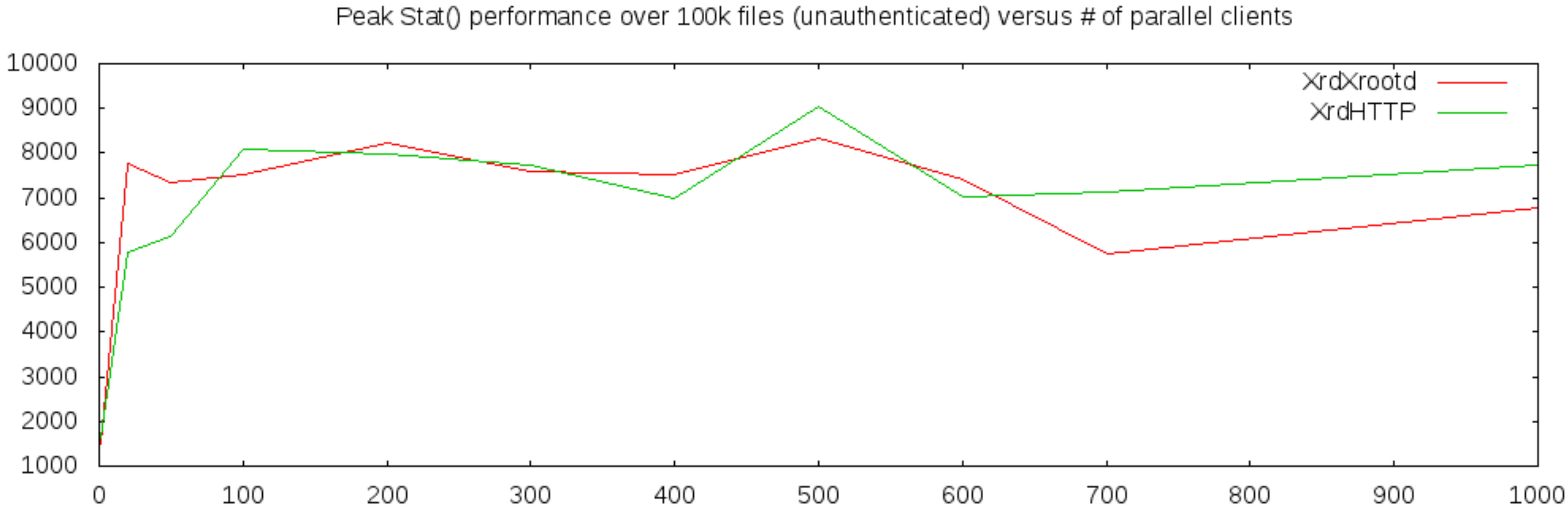
- **DPM**
- dCache
- STORM
- **HTTP for XRootD**
- **HTTP Federations**
- **FTS3**
- **DAVIX**
- **ROOT with TDavixFile**



- We want seamless interoperability, which is more than coexistence
- Performance and ability to compose services
- Any standard client will work and give its features
- Our contribution "fills the gaps" between plain Web and HEP data access
- Browsers are supported
- An advanced client will give all the features (implemented using standards)

A little benchmark

- No big differences xrootd/Http were expected. The xrootd framework handles all the raw data exchanges, threading and polling. Both proto implementations are efficient.
- For large files/chunks we see no difference, unless the data encryption is used (which the xrootd protocol does not support, but HTTPS does)
- For metadata operations we expected to see a slightly higher CPU usage (~5-10%), depending on the kind of pattern
- Actually seeing it or measuring its effect is not that straightforward, as in the following example, that was supposed to be a difficult test. They perform basically the same at 7K stats/sec



Conclusion

- XrdHTTP is our contribution for the Xrootd framework to be accessible with HTTP/WebDAV tools
  - Comes with Xrootd4
  - XrdHttpVOMS will come after the EPEL release of Xrootd4
- Opens new scenarios headed to compatibility
  - Interactive things, browsers, clouds, ...
- Preserves the existing features of the Xrootd framework
- Does not interfere with the activity done through the Xrootd protocol
  - Can share the same port 1094, no firewall changes, both protocols work together
- Low overhead
- Low config/management overhead